TLP **16** (5–6): 619–635, 2016. © Cambridge University Press 2016
doi:10.1017/S1471068416000478

*Query answering in resource-based answer set semantics**

STEFANIA COSTANTINI
DISIM, Università di L'Aquila
(*e-mail:* stefania.costantini@univaq.it)

ANDREA FORMISANO
DMI, Università di Perugia — GNCS-INdAM
(*e-mail:* formis@dmi.unipg.it)

submitted 6 May 2016; revised 8 July 2016; accepted 22 August 2016

Abstract

In recent work we defined resource-based answer set semantics, which is an extension to answer set semantics stemming from the study of its relationship with linear logic. In fact, the name of the new semantics comes from the fact that in the linear-logic formulation every literal (including negative ones) were considered as a resource. In this paper, we propose a query-answering procedure reminiscent of Prolog for answer set programs under this extended semantics as an extension of XSB-resolution for logic programs with negation.[1] We prove formal properties of the proposed procedure.
Under consideration for acceptance in TPLP.

KEYWORDS: Answer Set Programming, Procedural Semantics, Top-down Query-answering

1 Introduction

Answer set programming (ASP) is nowadays a well-established programming paradigm based on answer set semantics (Gelfond and Lifschitz 1988; Marek and Truszczyński 1999), with applications in many areas (cf., e.g., (Baral 2003; Truszczyński 2007; Gelfond 2007) and the references therein). Nevertheless, as noted in (Gebser *et al.* 2009; Bonatti *et al.* 2008), few attempts to construct a goal-oriented proof procedure exist, though there is a renewal of interest, as attested, e.g., by the recent work presented in (Marple and Gupta 2014). This is due to the very nature of the answer set semantics, where a program may admit none or several answer sets, and where the semantics enjoys no locality, or, better, no *Relevance* in the sense of (Dix 1995): no subset of the given program can in general be identified, from where the decision of atom A (intended as a goal, or query)

* This research is partially supported by YASMIN (RdB-UniPG2016/17) and FCRPG.2016.0105.021 projects.
[1] A preliminary shorter version of this paper appeared in (Costantini and Formisano 2014).

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

belonging or not to some answer set can be drawn. An incremental construction of approximations of answer sets is proposed in (Gebser *et al.* 2009) to provide a ground for local computations and top-down query answering. A sound and complete proof procedure is also provided. The approach of (Bonatti *et al.* 2008) is in the spirit of "traditional" SLD-resolution (Lloyd 1993), and can be used with non-ground queries and with non-ground, possibly infinite, programs. Soundness and completeness results are proven for large classes of programs. Another way to address the query-answering problem is discussed in (Lin and You 2002). This work describes a canonical rewriting system that turns out to be sound and complete under the *partial stable model semantics*. In principle, as the authors observe, the inference procedure could be completed to implement query-answering w.r.t. stable model semantics by circumventing the lack of Relevance. A substantially different approach to ASP computation is proposed in (Gebser and Schaub 2006) where the authors define a tableau-based framework for ASP. The main aim consists in providing a formal framework for characterizing inference operations and strategies in ASP-solvers. The approach is not based on query-oriented top-down evaluation, indeed, each branch in a tableau potentially corresponds to a computation of an answer set. However, one might foresee the possibility of exploiting such a tableau system to check answer set existence subject to query satisfaction.

A relevant issue concerning goal-oriented answer-set-based computation is related to sequences of queries. Assume that one would be able to pose a query $?\!- Q_1$ receiving an answer "yes", to signify that Q_1 is entailed by some answer set of the given program Π. Possibly, one might intend subsequent queries to be answered in the same *context*, i.e. a subsequent query $?\!- Q_2$ might ask whether some of the answer sets entailing Q_1 also entails Q_2. This might go on until the user explicitly "resets" the context. Such an issue, though reasonable in practical applications, has hardly been addressed up to now, due to the semantic difficulties that we have mentioned. A viable approach to these problems takes inspiration from the research on RASP (Resource-based ASP), which is a recent extension of ASP, obtained by explicitly introducing the notion of *resource* (Costantini and Formisano 2010). A RASP and linear-logic modeling of default negation as understood under the answer set semantics has been introduced in (Costantini and Formisano 2013). This led to the definition of an extension to the answer set semantics, called *Resource-based Answer Set Semantics* (RAS). The name of the new semantics comes from the fact that in the linear-logic formulation every literal (including negative ones) is considered as a resource that is "consumed" (and hence it becomes no more available) once used in a proof. This extension finds an alternative equivalent definition in a variation of the auto-epistemic logic characterization of answer set semantics discussed in (Marek and Truszczyński 1993).

We refer the reader to (Costantini and Formisano 2015) for a discussion of the new semantics from several points of view, and to (Costantini and Formisano 2016) for a summary of its formal definition. Under resource-based answer set semantics there are no inconsistent programs, i.e., every program admits (resource-based) answer set. Consider for instance the program $\Pi_1 = \{old \leftarrow not\ old\}$. Under the answer set semantics, Π_1 is inconsistent (has no answer sets) because it consists

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

of a unique odd cycle and no supported models exists. If we extend the program to $\Pi_2 = \{old \leftarrow not\ old.\ \ old \leftarrow not\ young.\}$ then $\{old\}$ is an answer set: in fact, the first rule is overridden by the second rule which allows *old* to be derived. Under the resource-based answer set semantics the first rule is ignored in the first place: in fact, Π_1 has a unique resource-based answer set which is the empty set. Intuitively, this results from interpreting default negation *not A* as "I assume that *A* is false" or, in autoepistemic terms (Marek and Truszczyński 1991a; Marek and Truszczyński 1991b) "I believe that I don't believe *A*". So, since deriving *A* accounts to denying the assumption of *not A*, such a derivation is disallowed as it would be contradictory. It is not considered to be inconsistent because default negation is not negation in classical logic: in fact, the attempt of deriving *A* from *not A* in classical logic leads to an inconsistency, while contradicting one's own assumption is (in our view) simply meaningless, so a rule such as the one in Π_1 is plainly ignored. Assume now to further enlarge the program, by obtaining $\Pi_3 = \{old \leftarrow not\ old.\ \ old \leftarrow not\ young.\ \ young \leftarrow old.\}$. There are again no answer sets, because by combining the last two rules a contradiction on *young* is determined, though indirectly. In resource-based answer set semantics there is still the answer set $\{old\}$, as the indirect contradiction is ignored: having assumed *not young* makes *young* unprovable.

In standard ASP, a constraint such as $\leftarrow L_1, \ldots, L_h$ where the L_is are literals is implemented by translating it into the rule $p \leftarrow not\ p, L_1, \ldots, L_h$ with *p* fresh atom. This is because, in order to make the contradiction on *p* harmless, one of the L_is must be false: otherwise, no answer set exists. Under resource-based answer set semantics such a transposition no longer works. Thus, constraints related to a given program are not seen as part of the program: rather, they must be defined separately and associated to the program. Since resource-based answer sets always exist, constraints will possibly exclude (a-posteriori) some of them. Thus, constraints act as a filter on resource-based answer sets, leaving those which are *admissible* with respect to given constraints.

In this paper we discuss a top-down proof procedure for the new semantics. The proposed procedure, beyond query-answering, also provides contextualization, via a form of tabling; i.e., a table is associated with the given program, and initialized prior to posing queries. Such table contains information useful for both the next and the subsequent queries. Under this procedure, $?\!- A$ (where we us assume with no loss of generality that *A* is an atom), succeeds whenever there exists some resource-based answer set *M* where $A \in M$. Contextualization implies that given a sequence of queries, for instance $?\!- A, ?\!- B$, both queries succeed if there exists some resource-based answer set *M* where $A \in M \wedge B \in M$: this at the condition of evaluating $?\!- B$ on the program table as left by $?\!- A$ (analogously for longer sequences). In case the table is reset, subsequent queries will be evaluated independently of previous ones. Success of $?\!- A$ must then be validated with respect to constraints; this issue is only introduced here, and will be treated in a future paper.

Differently from (Gebser *et al.* 2009), the proposed procedure does not require incremental answer set construction when answering a query and is not based on preliminary program analysis as done in (Marple and Gupta 2014). Rather, it

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

exploits the fact that resource-based answer set semantics enjoys the property of Relevance (Dix 1995) (whereas answer set semantics does not). This guarantees that the truth value of an atom can be established on the basis of the subprogram it depends upon, and thus allows for top-down computation starting from a query. For previous sample programs Π_2 and Π_3, query ?– *old* succeeds, while ?– *young* fails. W.r.t. the top-down procedure proposed in (Bonatti *et al.* 2008), we do not aim at managing function symbols (and thus programs with infinite grounding), so concerning this aspect our work is more limited.

As answer set semantics and resource-based answer set semantics extend the well-founded semantics (Van Gelder *et al.* 1991), we take as a starting point XSB-resolution (Swift and Warren 2012; Chen and Warren 1993), an efficient, fully described and implemented procedure which is correct and complete w.r.t. the well-founded semantics. In particular, we define RAS-XSB-resolution and discuss its properties; we prove correctness and completeness for every program (under the new semantics). We do not provide the full implementation details that we defer to a next step; in fact, this would imply suitably extending and reworking all operative aspects related to XSB. Thus, practical issues such as efficiency and optimization are not dealt with in the present paper and are rather deferred to future work of actual specification of an implementation. The proposed procedure is intended as a proof-of-concept rather than as an implementation guideline.

RAS-XSB resolution can be used for answer set programming under the software engineering discipline of dividing the program into a consistent "base" level and a "top" level including constraints. Therefore, even to readers not particularly interested in the new semantics, the paper proposes a full top-down query-answering procedure for ASP, though applicable under such (reasonable) limitation. In summary, RAS-XSB-Resolution:

- can be used for (credulous) top-down query-answering on logic programs under the resource-based answer set semantics and possibly under the answer set semantics, given the condition that constraints are defined separately from the "main" program;
- it is meant for the so-called "credulous reasoning" in the sense that given, say, query ?– A (where A is an atom), it determines whether there exists any (resource-based) answer set M such that $A \in M$;
- it provides "contextual" query-answering. It is possible to pose subsequent queries, say ?– $A_1, \ldots,$?– A_n; if they all succeed, there exists some (resource-based) answer set M such that $\{A_1, \ldots, A_n\} \subseteq M$; this extends to the case when only some of them succeed, where successful atoms are all in M and unsuccessful ones are not;
- does not require either preliminary program analysis or incremental answer-set construction, and does not impose any kind of limitation over the class of resource-based answer set programs which are considered (for answer set programs, there is the above-mentioned limitation on constraints).

This paper is organized as follows. After a presentation of resource-based answer set semantics in Section 2, we present the proposed query-answering procedure

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

in Section 3, and conclude in Section 4. In the rest of the paper, we refer to the standard definitions concerning propositional general logic programs and ASP (Lloyd 1993; Apt and Bol 1994; Gelfond 2007). If not differently specified, we implicitly refer to the *ground* version of a program Π. We do not consider "classical negation", double negation *not not A*, disjunctive programs, or the various useful programming constructs added over time to the basic ASP paradigm (Simons *et al.* 2002; Costantini and Formisano 2011; Faber *et al.* 2011).

2 Background on Resource-based ASP

The denomination "resource-based" answer set semantics (RAS) stems from the linear logic formulation of ASP proposed in (Costantini and Formisano 2013) which constituted the original inspiration for the new semantics. In this perspective, the negation *not A* of some atom *A* is considered to be a *resource* of unary amount, where:

- *not A* is *consumed* whenever it is used in a proof, thus preventing *A* to be proved, for retaining consistency;
- *not A* becomes no longer available whenever *A* is proved.

Consider for instance the following well-known sample answer set program consisting of a ternary odd cycle and concerning someone who wonders where to spend her vacation:

$$beach \leftarrow not\ mountain. \qquad mountain \leftarrow not\ travel. \qquad travel \leftarrow not\ beach.$$

In ASP, such program is inconsistent. Under the new semantics, there are three resource-based answer sets: $\{beach\}$, $\{mountain\}$, and $\{travel\}$. Take for instance the first one, $\{beach\}$. In order to derive the conclusion *beach* the first rule can be used; in doing so, the premise *not mountain* is consumed, thus disabling the possibility of proving *mountain*, which thus becomes false; *travel* is false as well, since it depends from a false premise.

We refer the reader to (Costantini and Formisano 2015) for a detailed discussion about logical foundations, motivations, properties, and complexity, and for examples of use. We provide therein characterizations of RAS in terms of linear logic, as a variation of the answer set semantics, and in terms of autoepistemic logic. Here we just recall that, due to the ability to cope with odd cycles, under RAS it is always possible to assign a truth value to all atoms: every program in fact admits at least one (possibly empty) resource-based answer set. A more significant example is the following (where, albeit in this paper we focus on the case of ground programs, for the sake of conciseness we make use of variables, as customary done to denote collections of ground literals/rules). The program models a recommender agent, which provides a user with indication to where it is possible to spend the evening, and how the user should dress for such an occasion. The system is also able to take user preferences into account.

The resource-based answer set program which constitutes the core of the system is the following. There are two ternary cycles. The first one specifies that a person can be dressed either formally or normally or in an eccentric way. Only old-fashioned

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

persons dress formally, and only persons with a young mind dress in an eccentric way. Later on, it is stated by two even cycles that any person can be old-fashioned or young-minded, independently of the age that, by the second odd cycle, can be young, middle, or old. The two even cycles interact, so that only one option can be taken. Then, it is stated that one is admitted to an elegant restaurant if (s)he is formally dressed, and to a disco if (s)he is dressed in an eccentric way. To spend the evening either in an elegant restaurant or in a disco one must be admitted. Going out in this context means either going to an elegant restaurant (for middle-aged or old people) or to the disco for young people, or sightseeing for anyone.

$$formal\_dress(P) \leftarrow person(P), not\ normal\_dress(P), old\_fashioned(P).$$
$$normal\_dress(P) \leftarrow person(P), not\ eccentric\_dress(P).$$
$$eccentric\_dress(P) \leftarrow person(P), not\ formal\_dress(P), young\_mind(P).$$
$$old(P) \leftarrow person(P), not\ middleaged(P).$$
$$middleaged(P) \leftarrow person(P), not\ young(P).$$
$$young(P) \leftarrow person(P), not\ old(P).$$
$$old\_fashioned(P) \leftarrow person(P), not\ young\_mind(P), not\ noof(P).$$
$$noof(P) \leftarrow person(P), not\ old\_fashioned(P).$$
$$young\_mind(P) \leftarrow person(P), not\ old\_fashioned(P), not\ noym(P).$$
$$noym(P) \leftarrow person(P), not\ young\_mind(P).$$
$$admitted\_elegant\_restaurant(P) \leftarrow person(P), formal\_dress(P).$$
$$admitted\_disco(P) \leftarrow person(P), eccentric\_dress(P).$$
$$go\_disco(P) \leftarrow person(P), young(P), admitted\_disco(P).$$
$$go\_elegant\_restaurant(P) \leftarrow person(P), admitted\_elegant\_restaurant(P).$$
$$go\_elegant\_restaurant(P) \leftarrow person(P), middleaged(P), admitted\_elegant\_restaurant(P).$$
$$go\_sightseeing(P) \leftarrow person(P).$$
$$go\_out(P) \leftarrow middleaged(P), go\_elegant\_restaurant(P).$$
$$go\_out(P) \leftarrow old(P), go\_elegant\_restaurant(P).$$
$$go\_out(P) \leftarrow young(P), go\_disco(P).$$
$$go\_out(P) \leftarrow go\_sightseeing(P).$$

The above program, if considered as an answer set program, has a (unique) empty resource-based answer set, as there are no facts (in particular there are no facts for the predicate *person* to provide values for the placeholder P). Now assume that the above program is incorporated into an interface system which interacts with a user, say George, who wants to go out and wishes to be made aware of his options. The system may thus add the fact $person(george)$ to the program. While, in ASP the program would become inconsistent, in RASP the system would, without any more information, advise George to go sightseeing. This is, in fact, the only advice that can be extracted from the unique resource-based answer set of the resulting program. If the system might obtain or elicit George's age, the options would be many more, according to the hypotheses about him being old-fashioned or young-minded. Moreover, for each option (except sightseeing) the system would be able to extract the required dress code. George might want to express a preference, e.g., going to the disco. Then the system might add to the program the rule

$$preference(P) \leftarrow person(P),\ go\_disco(P).$$

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

and state the constraint $\leftarrow$ *not preference*(P) that "forces" the preference to be satisfied, thus making George aware of the hypotheses and conditions under which he might actually go to the disco. Namely, they correspond to the unique resource-based answer set where George is young, young-minded and dresses in an eccentric way.

However, in resource-based answer set semantics constraints cannot be modeled (as done in ASP) as "syntactic sugar", in terms of unary odd cycles involving fresh atoms. Hence, they have to be modeled explicitly. Without loss of generality, we assume, from now on, the following simplification concerning constraints. Each constraint $\leftarrow L_1,\ldots,L_k$, where each L_i is a literal, can be rephrased as simple constraint $\leftarrow H$, where H is a fresh atom, plus rule $H \leftarrow L_1,\ldots,L_k$ to be added to the given program Π. So, H occurs in the set S_Π of all the atoms of Π.

Definition 2.1
Let Π be a program and $\mathscr{C} = \{\mathscr{C}_1,\ldots,\mathscr{C}_k\}$ be a set of constraints, each $\mathscr{C}_i$ in the form $\leftarrow H_i$.

- A resource-based answer set M for Π is *admissible* w.r.t. $\mathscr{C}$ if for all $i \leqslant k$ $H_i \notin M$.
- The program Π is called "admissible" w.r.t. $\mathscr{C}$ if it has an admissible answer w.r.t. $\mathscr{C}$.

It is useful for what follows to evaluate RAS with respect to general properties of semantics of logic programs introduced in (Dix 1995), that we recall below. A semantic SEM for logic programs is intended as a function which associates a logic program with a set of sets of atoms, which constitute the intended meaning.

Definition 2.2
Given any semantics SEM and a ground program Π, *Relevance* states that for all literals L it holds that $SEM(\Pi)(L) = SEM(rel\_rul(\Pi; L))(L)$.

Relevance implies that the truth value of any literal under that semantics in a given program, is determined solely by the subprogram consisting of the relevant rules. The answer set semantics does not enjoy Relevance (Dix 1995). This is one reason for the lack of goal-oriented proof procedures. Instead, it is easy to see that resource-based answer set semantics enjoys Relevance. Resource-based answer set semantics, like most semantics for logic programs with negation, enjoys *Reduction*, which simply assures that the atoms not occurring in the heads of a program are always assigned truth value false. Another important property is *Modularity*, defined in (Dix 1995) as follows (where the reduct Π^M of program Π w.r.t. set of atoms M):

Definition 2.3
Given any semantics SEM, a ground program Π let $\Pi = \Pi_1 \cup \Pi_2$ where for every atom A occurring in Π_2, $rel\_rul(\Pi; A) \subseteq \Pi_2$. We say that SEM enjoys *Modularity* if it holds that $SEM(\Pi) = SEM(\Pi_1^{SEM(\Pi_2)} \cup \Pi_2)$.

If Modularity holds, then the semantics can be always computed by splitting a program in its subprograms (w.r.t. relevant rules). Intuitively, in the above definition,

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

the semantics of Π_2, which is self-contained, is first computed. Then, the semantics of the whole program can be determined by reducing Π_1 w.r.t. $SEM(\Pi_2)$. We can state (as a consequence of Relevance) that resource-based answer set semantics enjoys Modularity.

Proposition 2.1
Given a ground program Π let $\Pi = \Pi_1 \cup \Pi_2$, where for every atom A occurring in Π_2, $rel\_rul(\Pi; A) \subseteq \Pi_2$. A set M of atoms is a resource-based answer set of Π iff there exists a resource-based answer set S of Π_2 such that M is a resource-based answer set of $\Pi_1^S \cup \Pi_2$.

Modularity also impacts on constraint checking, i.e. on the check of admissibility of resource-based answer sets. Considering, in fact, a set of constraints $\{\mathscr{C}_1, \ldots, \mathscr{C}_n\}$, $n > 0$, each $\mathscr{C}_i$ in the form $\leftarrow H_i$, and letting for each $i \leqslant n$ $rel\_rul(\Pi; H_i) \subseteq \Pi_2$, from Proposition 2.1 it follows that, if a resource-based answer set X of Π_2 is admissible (in terms of Definition 2.1) w.r.t. $\{\mathscr{C}_1, \ldots, \mathscr{C}_n\}$, then any resource-based answer set M of Π such that $X \subseteq M$ is also admissible w.r.t. $\{\mathscr{C}_1, \ldots, \mathscr{C}_n\}$. In particular, Π_2 can be identified in relation to a certain query:

Definition 2.4
Given a program Π, a constraint $\leftarrow H$ associated to Π is *relevant for query* $?\!- A$ if $rel\_rul(\Pi; A) \subseteq rel\_rul(\Pi; H)$.

3 A Top-down Proof Procedure for RAS

As it is well-known, the answer set semantics extends the well-founded semantics (wfs) (Van Gelder *et al.* 1991) that provides a unique three-valued model $\langle W^+, W^- \rangle$, where atoms in W^+ are *true*, those in W^- are *false*, and all the others are *undefined*. In fact, the answer set semantics assigns, for consistent programs truth values to the undefined atoms. However, the program can be *inconsistent* because of odd cyclic dependencies. The improvement of resource-based answer set semantics over the answer set semantics relies exactly on its ability to deal with odd cycles that the answer set semantics interprets as inconsistencies. So, in any reasonable potential query-answering device for ASP, a query $?\!- A$ to an ASP program Π may be reasonably expected to succeed or fail if A belongs to W^+ or W^-, respectively. Such a procedure will then be characterized according to how to provide an answer when A is undefined under the wfs.

An additional problem with answer set semantics is that query $?\!- A$ might *locally* succeed, but still, for the lack of Relevance, the overall program may not have answer sets. In resource-based answer set semantics instead, every program has one or more resource-based answer set: each of them taken singularly is then admissible or not w.r.t. the integrity constraints. This allows one to perform constraint checking upon success of query $?\!- A$.

We will now define the foundations of a top-down proof procedure for resource-based answer set semantics, which we call RAS-XSB-resolution. The procedure has to deal with atoms involved in negative circularities, that must be assigned a truth

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

value according to some resource-based answer set. We build upon XSB-resolution, for which an ample literature exists, from the seminal work in (Chen and Warren 1993) to the most recent work in (Swift and Warren 2012) where many useful references can also be found. For lack of space XSB-resolution is not described here. XSB in its basic version, XOLDTNF-resolution (Chen and Warren 1993) is shortly described in (Costantini and Formisano 2016). We take for granted basic notions concerning proof procedures for logic programming, such as for instance backtracking. For the relevant definitions we refer to (Lloyd 1993). Some notions are however required here for the understanding of what follows. In particular, it is necessary to illustrate detection of cycles on negation.

Definition 3.1 (*XSB Negative Cycles Detection*)

- Each call to atom A has an associated set N of negative literals, called the *negative context* for A, so the call takes the form (A, N).
- Whenever a negative literal $not\ B$ is selected during the evaluation of some A, there are two possibilities: (i) $not\ B \notin N$: this will lead to the call $(B, N \cup \{not\ B\})$; (ii) $not\ B \in N$, then there is a possible negative loop, and B is called a *possibly looping negative literal*.
- For the initial call of any atom A, N is set to empty.

In order to assume that a literal $not\ B$ is a looping negative literal, that in XSB assumes truth value *undefined*, the evaluation of B must however be *completed*, i.e. the search space must have been fully explored without finding conditions for success or failure.

Like in XSB, for each program Π a table $\mathcal{T}(\Pi)$ records useful information about proofs. As a small extension w.r.t. XSB-Resolution, we record in $\mathcal{T}(\Pi)$ not only successes, but also failures. XSB-resolution is, for Datalog programs, correct and complete w.r.t. the wfs. Thus, it is useful to state the following definition.

Definition 3.2

Given a program Π and an atom A, we say that

- *A definitely succeeds* iff it succeeds via XSB- (or, equivalently, XOLDTNF-) resolution, and thus A is recorded in $\mathcal{T}(\Pi)$ with truth value *true*. For simplicity, we assume A occurs in $\mathcal{T}(\Pi)$.
- *A definitely fails* iff it fails via XSB- (or, equivalently, XOLDTNF-) resolution, and thus A is recorded in $\mathcal{T}(\Pi)$ with truth value *false*. For simplicity, we assume $not\ A$ occurs in $\mathcal{T}(\Pi)$.

To represent the notion of negation as a resource, we initialize the program table prior to posing queries and we manage the table during a proof so as to state that:

- the negation of any atom which is not a fact is available unless this atom has been proved;
- the negation of an atom which has been proved becomes unavailable;
- the negationof an atom which cannot be proved is always available.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

Definition 3.3 (*Table Initialization in RAS-XSB-resolution*)
Given a program Π and an associated table $\mathcal{T}(\Pi)$, *Initialization* of $\mathcal{T}(\Pi)$ is performed by inserting, for each atom A occurring as the conclusion of some rule in Π, a fact *yesA* (where *yesA* is a fresh atom).

The meaning of *yesA* is that the negation *not A* of A has not been proved. If *yesA* is in the table, then A can possibly succeed. Success of A "absorbs" *yesA* and prevents *not A* from succeeding. Failure of A or success of *not A* "absorbs" *yesA* as well, but *not A* is asserted. $\mathcal{T}(\Pi)$ will in fact evolve during a proof into subsequent states, as specified below.

Definition 3.4 (*Table Update in RAS-XSB-resolution*)
Given a program Π and an associated table $\mathcal{T}(\Pi)$, referring to the definition of RAS-XSB-resolution (cf. Definition 3.5 below), the table update is performed as follows.

- Upon success of subgoal A, *yesA* is removed from $\mathcal{T}(\Pi)$ and A is added to $\mathcal{T}(\Pi)$.
- Upon failure of subgoal A, *yesA* is removed from $\mathcal{T}(\Pi)$ and *not A* is added to $\mathcal{T}(\Pi)$.
- Upon success of subgoal *not A*, *yesA* is removed from $\mathcal{T}(\Pi)$ and *not A* is added to $\mathcal{T}(\Pi)$. However:
  - if *not A* succeeds by case 3.b, then such modification is permanent;
  - if *not A* succeeds either by case 3.c or by case 3.d, then in case of failure of the parent subgoal the modification is retracted, i.e. *yesA* is restored in $\mathcal{T}(\Pi)$ and *not A* is removed from $\mathcal{T}(\Pi)$.

We refer the reader to the examples provided below for a clarification of the table-update mechanism. In the following, without loss of generality we can assume that a query is of the form ?– A, where A is an atom. Success or failure of this query is established as follows. Like in XSB-resolution, we assume that the call to query A implicitly corresponds to the call (A, N) where N is the negative context of A, which is initialized to $\emptyset$ and treated as stated in Definition 3.1.

Definition 3.5 (*Success and failure in RAS-XSB-resolution*)
Given a program Π and its associated table $\mathcal{T}(\Pi)$, notions of success and failure and of modifications to $\mathcal{T}(\Pi)$ are extended as follows with respect to XSB-resolution.

(1) Atom A succeeds iff *yesA* is in $\mathcal{T}(\Pi)$, and one of the following conditions holds.
   (a) A definitely succeeds (which includes the case where A is present in $\mathcal{T}(\Pi)$).
   (b) There exists in Π either fact A or a rule of the form $A \leftarrow L_1, \ldots, L_n$, $n > 0$, such that neither A nor *not A* occur in the body and every literal L_i, $i \leqslant n$, succeeds.

(2) Atom A fails iff one of the following conditions holds.
   (a) *yesA* is not present in $\mathcal{T}(\Pi)$.
   (b) A definitely fails.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

(c) There is no rule of the form $A \leftarrow L_1, \ldots, L_n$, $n > 0$, such that every literal L_i succeeds.

(3) Literal *not A* succeeds if one of the following is the case:
   (a) *not A* is present in $\mathscr{T}(\Pi)$.
   (b) *A* fails.
   (c) *not A* is *allowed to succeed*.
   (d) *A* is *forced to failure*.

(4) Literal *not A* fails if *A* succeeds.

(5) *not A* is *allowed to succeed* whenever the call $(A, \emptyset)$ results, whatever sequence of derivation steps is attempted, in the call $(A, N \cup \{not\ A\})$. I.e., the derivation of *not A* incurs through layers of negation again into *not A*.

(6) *A* is *forced to failure* when the call $(A, \emptyset)$ always results in the call $(A, \{not\ A\})$, whatever sequence of derivation steps is attempted. I.e., the derivation of *not A* incurs in *not A* directly.

From the above extension of the notions of success and failure we obtain RAS-XSB-resolution as an extended XSB-resolution. Actually, in the definition we exploit XSB (or, more precisely, XOLDTNF), as a "plugin" for definite success and failure, and we add cases which manage subgoals with answer *undefined* under XSB. This is not exactly ideal from an implementation point of view. In future work, we intend to proceed to a much more effective integration of XSB with the new aspects that we have introduced, and to consider efficiency and optimization issues that are presently neglected.

Notice that the distinction between RAS-XSB-resolution and XSB-resolution is determined by cases 3.c and 3.d of Definition 3.5, which manage literals involved in negative cycles. The notions of *allowance to succeed* (case 5) and of *forcing to failure* (case 6) are crucial. Let us illustrate the various cases via simple examples:

- Case 3.c deals with literals depending negatively upon themselves through other negations. Such literals can be assumed as *hypotheses*. Consider, for example, the program $a \leftarrow not\ b.\ \ b \leftarrow not\ a.$ Query $?\!-a$ succeeds by assuming *not b*, which is correct w.r.t. (resource-based) answer set $\{a\}$. If, however, the program is $a \leftarrow not\ b, not\ e.\ \ b \leftarrow not\ a.\ \ e.$ then, the same query $?\!-a$ fails upon definite failure of *not e*, so the hypothesis *not b* must be retracted. This is, in fact, stated in the specification of table update (Definition 3.4).
- Case 3.d deals with literals depending negatively upon themselves directly. Such literals can be assumed as *hypotheses*. Consider, for example, the program $p \leftarrow a.\ \ a \leftarrow not\ p.$. Query $?\!-a$ succeeds because the attempt to prove *not p* comes across *not p* (through *a*), and thus *p* is forced to failure. This is correct w.r.t. resource-based answer set $\{a\}$. Notice that for atoms involved in negative cycles the positive-cycle detection is relaxed, as some atom in the cycle will either fail or been forced to failure. If however the program is $p \leftarrow a.\ \ a \leftarrow not\ p, not\ q.$ then, the same query $?\!-a$ fails upon definite failure of *not q*, so the hypothesis *not p* must be retracted. This is in fact stated in the specification of table update (Definition 3.4).

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

We provide below a high-level definition of the overall proof procedure (overlooking implementation details), which resembles plain SLD-resolution.

Definition 3.6 (*A naive RAS-XSB-resolution*)
Given a program Π, let assume as input the data structure $\mathscr{T}(\Pi)$ used by the proof procedure for tabling purposes, i.e. the table associated with the program. Given a query $?\!- A$, the list of current subgoals is initially set to $\mathscr{L}_1 = \{A\}$. If in the construction of a proof-tree for $?\!- A$ a literal L_{i_j} is selected in the list of current subgoals $\mathscr{L}_i$, we have that: if L_{i_j} succeeds then we take L_{i_j} as proved and proceed to prove $L_{i_{j+1}}$ after the related updates to the program table. Otherwise, we have to backtrack to the previous list $\mathscr{L}_{i-1}$ of subgoals.

Conditions for success and failure are those specified in Definition 3.5. Success and failure determine the modifications to $\mathscr{T}(\Pi)$ specified in Definition 3.4. Backtracking does not involve restoring previous contents of $\mathscr{T}(\Pi)$, as subgoals which have been proved can be usefully employed as lemmas. In fact, the table is updated only when the entire search space for a subgoal has been explored. The only exception concerns negative subgoals which correspond to literals involved in cycles: in fact, they are to considered as hypotheses that could later be retracted. For instance, consider the program

$$q \leftarrow not\ a, c. \qquad q \leftarrow not\ b. \qquad a \leftarrow not\ b. \qquad b \leftarrow not\ a.$$

and query $?\!- q$. Let us assume clauses are selected in the order. So, the first clause for q is selected, and *not a* is initially allowed to succeed (though involved in a negative cycle with *not b*). However, upon failure of subgoal c with consequent backtracking to the second rule for q, lemma *not A* must be retracted from the table: this in fact enables *not b* to be allowed to succeed, so determining success of the query.

Definition 3.7
Given a program Π and its associated table $\mathscr{T}(\Pi)$, a *free query* is a query $?\!- A$ which is posed on Π when the table has just been initialized. A *contextual query* is a query $?\!- B$ which is posed on Π leaving the associated table in the state determined by former queries.

Success of query $?\!- A$ means (as proved in Theorem 3.1 below) that there exist resource-based answer sets that contain A. The final content of $\mathscr{T}(\Pi)$ specifies literals that hold in these sets (including A). Precisely, the state of $\mathscr{T}(\Pi)$ characterizes a set $\mathscr{S}_{\mathscr{T}(\Pi)_A}$ resource-based answer sets of Π, such that for all $M \in \mathscr{S}_{\mathscr{T}(\Pi)_A}$, and for every atom D, $D \in \mathscr{T}(\Pi)$ implies $D \in M$ and $not\ D \in \mathscr{T}(\Pi)$ implies $D \notin M$. Backtracking on $?\!- A$ accounts to asking whether there are other different resource-based answer sets containing A, and implies making different assumptions about cycles by retracting literals which had been assumed to succeed. Instead, posing a subsequent query $?\!- B$ without resetting the contents of $\mathscr{T}(\Pi)$, which constitutes a *context*, accounts to asking whether some of the answer sets in $\mathscr{S}_{\mathscr{T}(\Pi)_A}$ also contain B. Posing such a contextual query, the resulting table reduces previously-identified resource-based answer sets to a possibly smaller set $\mathscr{S}_{\mathscr{T}(\Pi)_{A\cup B}}$ whose elements include both A and B (see Theorem 3.2 below). Contextual queries and sequences of contextual queries are formally defined below.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

Definition 3.8 (*Query sequence*)
Given a program Π and $k > 1$ queries $?\!- A_1, \ldots, ?\!- A_k$ performed one after the other, assume that $\mathscr{T}(\Pi)$ is initialized only before posing $?\!- A_1$. Then, $?\!- A_1$ is a free query where each $?\!- A_i$, is a *contextual query*, evaluated w.r.t. the previous ones.

To show the application of RAS-XSB-resolution to single queries and to a query sequence, let us consider the sample following program Π, which includes virtually all cases of potential success and failure. The well-founded model of this program is $\langle\{e\},\{d\}\rangle$ while the resource-based answer sets are $M_1 = \{a,e,f,h,s\}$ and $M_2 = \{e,h,g,s\}$.

r_1. $a \leftarrow not\ g.$ r_3. $s \leftarrow not\ p.$ r_5. $h \leftarrow not\ p.$ r_7. $f \leftarrow not\ g, e.$
r_2. $g \leftarrow not\ a.$ r_4. $p \leftarrow h.$ r_6. $f \leftarrow not\ a, d.$ r_8. $e.$

Initially, $\mathscr{T}(\Pi)$ includes *yesA* for every atom occurring in some rule head: $\mathscr{T}(\Pi) = \{yesa, yesb, yesc, yese, yesf, yesg, yesp, yesh, yess\}$. Below we illustrate some derivations. We assume that applicable rules are considered from first (r_1) to last (r_8) as they are ordered in the program, and literals in rule bodies from left to right.

Let us first illustrate the proof of query $?\!- f$. Each additional layer of $?\!-$ indicates nested derivation of A whenever literal *not A* is encountered. In the comment, we refer to cases of RAS-XSB-resolution as specified in Definition 3.5. Let us first consider query $?\!- f$.

$?\!- f.$
$?\!- not\ a, d.$ % via r_6
Subgoal *not a* is treated as follows.
$?\!-?\!- a.$
$?\!-?\!- not\ g.$ % via r_1
$?\!-?\!-?\!- g.$
$?\!-?\!-?\!- not\ a.$ % via r_2. *not a* succeeds by case 3.c, $\mathscr{T}(\Pi) = \mathscr{T}(\Pi) \cup \{not\ a\} \setminus \{yesa\}$
Subgoal d gives now rise to the following derivation.
$?\!- d.$ % d fails by case 2.b, so the parent goal f fails.
Backtracking is however possible, as there exists a second rule for f.
$?\!- not\ g, e.$ % via r_7
$?\!-?\!- g.$
$?\!-?\!- not\ a.$ % via r_2
$?\!-?\!-?\!- a.$
$?\!-?\!-?\!- not\ g.$ % via r_1.Thus, *not g* succeeds by case 3.c.$\mathscr{T}(\Pi) = \mathscr{T}(\Pi) \cup \{not\ g\} \setminus \{yesg\}$
Now, the second subgoal e remains to be completed:
$?\!- e.$ % e succeeds by case 1.b, and the overall query f succeeds by case 1.b.
$\mathscr{T}(\Pi) = \mathscr{T}(\Pi) \cup \{e, f\} \setminus \{yese, yesf\}$

Assuming now to go on to query the same context, i.e. without re-initializing $\mathscr{T}(\Pi)$, query $?\!- g$ quickly fails by case 2.a since $not\ g \in \mathscr{T}(\Pi)$. Query $?\!- e$ succeeds immediately by case 1.a as $e \in \mathscr{T}(\Pi)$. We can see that the context we are within corresponds to resource-based answer set M_1. Notice that, if resetting the context, $?\!- g$ would instead succeed as by case 1.b as *not a* can be allowed to succeed by case 3.c. Finally, a derivation for $?\!- s$ is obtained as follows:

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

?– s.
?– *not* p. % via r_3
?–?– p.
?–?– h. % via r_4
?–?– *not* p. % via r_5, *not* p succeeds by case 3.d, and p is forced to failure $\mathscr{T}(\Pi) = \mathscr{T}(\Pi) \cup \{not\ p\} \setminus \{yesp, yesh\}$.

Then, at the upper level, s and h succeed by case 1.b, and $\mathscr{T}(\Pi) \cup \{s\} \setminus \{yess\}$. Notice that forcing p to failure determines *not* p to succeed, and consequently allows h to succeed (where h is undefined under the wfs). The derivation of h involves the tricky case of a positive dependency through negation.

3.1 Properties of RAS-XSB-resolution

Properties of resource-based answer set semantics are strictly related to properties of RAS-XSB-resolution. In fact, thanks to Relevance we have soundness and completeness, and Modularity allows for contextual query and locality in constraint-checking. Such properties are summarized in the following Theorems (whose proofs can be found in (Costantini and Formisano 2016)).

Theorem 3.1
RAS-XSB-resolution is correct and complete w.r.t. resource-based answer set semantics, in the sense that, given a program Π, a query ?– A succeeds under RAS-XSB-resolution with an initialized $\mathscr{T}(\Pi)$ iff there exists resource-based answer set M for Π where $A \in M$.

Theorem 3.2
RAS-XSB-resolution is contextually correct and complete w.r.t. resource Answer Set semantics, in the sense that, given a program Π and a query sequence ?– $A_1, \ldots,$?– A_k, $k > 1$, where $\{A_1, \ldots, A_k\} \subseteq S_\Pi$ (i.e. the A_is are atoms occurring in Π), we have that, for $\{B_1, \ldots, B_r\} \subseteq \{A_1, \ldots, A_k\}$ and $\{D_1, \ldots, D_s\} \subseteq \{A_1, \ldots, A_k\}$, the queries ?– $B_1, \ldots,$?– B_r succeed while ?– $D_1, \ldots,$?– D_s fail under RAS-XSB-resolution, iff there exists resource-based answer set M for Π where $\{B_1, \ldots, B_r\} \subseteq M$ and $\{D_1, \ldots, D_s\} \cap M = \emptyset$.

This result extends immediately to queries including negative literals such as *not* H, $H \in S_\Pi$. We say that a query sequence contextually succeeds if each of the involved queries succeeds in the context (table) left by all former ones.

We defer a discussion of constraint checking to a future paper. Notice only that, given an admissible program Π and a constraint $\leftarrow C$ (where C is an atom), success of the query ?– *not* C in a certain context (given by $\mathscr{T}(\Pi)$) means that this constraint is fulfilled in the admissible resource-based answer sets Π selected by that context. If the context where ?– *not* C is executed results from a query ?– A, this implies by Theorem 3.2 that $\leftarrow C$ is fulfilled at least one admissible resource-based answer set including A. So, in admissible programs one should identify and check (a posteriori) constraints that are *relevant* to the query according to Definition 2.4.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

4 Concluding Remarks

A relevant question about RAS-XSB-resolution is whether it might be applicable to non-ground queries and programs. By resorting to standard unification, non-ground queries on ground programs can be easily managed. In future work we intend however to extend the procedure to non-ground programs without requiring preliminary program grounding. This should be made possible by the tabling mechanism, which stores ground positive and negative intermediate results, and by Relevance and Modularity of resource-based answer set semantics.

An important issue is whether RAS-XSB-resolution might be extended to plain ASP. Unfortunately, ASP programs may have a quite complicated structure: the effort of (Gebser *et al.* 2009) has been, in fact, that of performing a layer-based computation upon some conditions. Many answer set programs concerning real applications are however already expressed with constraints at the top layer, as required by our approach.

A comparison with existing proof procedures can be only partial, as these procedures cope with any answer set program, with its involved internal structure. So, overall our procedure imposes less 'a priori' conditions and has a simple definition, but this is obtained by means of a strong preliminary assumption about constraints. However, as the expressive power and complexity remain the same, our approach might constitute a way of simplifying implementation aspects without significant losses in "practical" expressiveness.

We intend to investigate an integration of RAS-XSB-resolution with principles and techniques introduced in (Bonatti *et al.* 2008), so as to further enlarge its applicability to what they call *finitary programs*, which are a large class of non-ground programs with function symbols. In fact, this approach allows programmers to make use of popular recursive definitions which are common in Prolog, and makes ASP technology even more competitive with respect to other state-of-the-art techniques.

In summary, we have proposed the theoretical foundations of a proof procedure related to a reasonable extension of answer set programming. The procedure has been obtained by taking as a basis XSB-resolution and its tabling features. Future work includes a precise design of a RAS-XSB-resolution implementation. Our objective is to realize an efficient inference engine, that should then be checked and experimented on (suitable versions of) well-established benchmarks (see, e.g., (Calimeri *et al.* 2016)). We intend in this sense to seek an integration with XSB, and with well-established ASP-related systems (cf. the discussion in (Giunchiglia *et al.* 2008)), already used for the implementation of the procedure proposed in (Bonatti *et al.* 2008).

Supplementary materials

For supplementary material for this article, please visit http://dx.doi.org/10.1017/S1471068416000478

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

References

Apt, K. R. and Bol, R. N. 1994. Logic programming and negation: A survey. *J. Log. Prog. 19/20*, 9–71.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, New York, NY, USA.

Bonatti, P. A., Pontelli, E. and Son, T. C. 2008. Credulous resolution for answer set programming. In *Proc. of AAAI 2008*, D. Fox and C. P. Gomes, Eds. AAAI Press, 418–423.

Calimeri, F., Gebser, M., Maratea, M. and Ricca, F. 2016. Design and results of the fifth answer set programming competition. *Artif. Intell. 231*, 151–181.

Chen, W. and Warren, D. S. 1993. A goal-oriented approach to computing the well-founded semantics. *J. Log. Prog. 17,* 2/3&4, 279–300.

Costantini, S. and Formisano, A. 2010. Answer set programming with resources. *J. of Logic and Computation 20,* 2, 533–571.

Costantini, S. and Formisano, A. 2011. Weight constraints with preferences in ASP. In *Proc. of LPNMR'11*. LNCS, vol. 6645. Springer, Vancouver, Canada, 229–235.

Costantini, S. and Formisano, A. 2013. RASP and ASP as a fragment of linear logic. *J. of Applied Non-Classical Logics 23,* 1-2, 49–74.

Costantini, S. and Formisano, A. 2014. Query answering in resource-based answer set semantics. In *Proc. of the 29th Italian Conference on Computational Logic*. CEUR, Torino, Italy. Also appeared in the 7th Workshop ASPOCP 2014.

Costantini, S. and Formisano, A. 2015. Negation as a resource: a novel view on answer set semantics. *Fundam. Inform. 140,* 3-4, 279–305.

Costantini, S. and Formisano, A. 2016. Online supplementary materials for "Query Answering in Resource-Based Answer Set Semantics". *TPLP archives*. See also the arXiv version in `http://arxiv.org/abs/1608.01604`, CoRR.

Dix, J. 1995. A classification theory of semantics of normal logic programs I-II. *Fundam. Inform. 22,* 3, 227–255 and 257–288.

Faber, W., Leone, N. and Pfeifer, G. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1, 278–298.

Gebser, M., Gharib, M., Mercer, R. E. and Schaub, T. 2009. Monotonic answer set programming. *J. Log. Comput. 19,* 4, 539–564.

Gebser, M. and Schaub, T. 2006. Tableau calculi for answer set programming. In *Proc. of ICLP 2006*, S. Etalle and M. Truszczyński, Eds. LNCS, vol. 4079. Springer, Seattle, USA, 11–25.

Gelfond, M. 2007. Answer sets. In *Handbook of Knowledge Representation. Chapter 7*. Elsevier, Amsterdam, The Netherlands, 285–316.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th Intl. Conf. and Symposium on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, Seattle, USA, 1070–1080.

Giunchiglia, E., Leone, N. and Maratea, M. 2008. On the relation among answer set solvers. *Ann. Math. Artif. Intell. 53,* 1-4, 169–204.

Lin, F. and You, J. 2002. Abduction in logic programming: A new definition and an abductive procedure based on rewriting. *Artificial Intelligence 140,* 1/2, 175–205.

Lloyd, J. W. 1993. *Foundations of Logic Programming*, 2nd ed. Springer, New York, USA.

Marek, V. W. and Truszczyński, M. 1991a. Autoepistemic logic. *J. of the ACM 38,* 3, 587–618.

Marek, V. W. and Truszczyński, M. 1991b. Computing intersection of autoepistemic expansions. In *Proc. LPNMR 1991*. MIT Press, Washington, D.C., USA, 35–70.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press

Marek, V. W. and Truszczyński, M. 1993. Reflective autoepistemic logic and logic programming. In *Proc. of LPNMR 1993*, A.Nerode and L.M.Pereira, Eds. The MIT Press, 115–131.

Marek, V. W. and Truszczyński, M. 1999. *Stable logic programming - an alternative logic programming paradigm*. Springer, Berlin, Heidelberg, 375–398.

Marple, K. and Gupta, G. 2014. Dynamic consistency checking in goal-directed answer set programming. *Theory and Practice of Logic Programming 14,* 4-5, 415–427.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

Swift, T. and Warren, D. S. 2012. XSB: Extending Prolog with tabled logic programming. *Theory and Practice of Logic Programming 12,* 1-2, 157–187.

Truszczyński, M. 2007. Logic programming for knowledge representation. In *Logic Programming, 23rd Intl. Conference, ICLP 2007*, V. Dahl and I. Niemelä, Eds. Springer, 76–88.

Van Gelder, A., Ross, K. A. and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.

https://doi.org/10.1017/S1471068416000478 Published online by Cambridge University Press